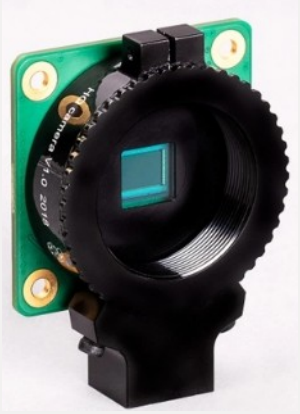
Part 06
-
Activating PiCamera

Version: 2021-012-29

Raspberry Pi Camera modules models

The Raspberry Pi Camera Modules are official products from the Raspberry Pi Foundation. The original 5-megapixel model (V1.3) was released in 2013, and an 8-megapixel Camera Module (V2.1) was released in 2016. For both iterations, there are visible light and infrared versions. A 12-megapixel High Quality Camera was released in 2020. There is no infrared version of the HQ Camera, however the IR Filter can be removed if required.

	Camera Module v1	Camera Module v2	HQ Camera
			
Size	25 × 24 × 9 mm	25 × 24 × 9 mm	38 x 38 x 18.4mm
Still resolution	5 Megapixels	8 Megapixels	12.3 Megapixels
Video modes	1080p30 720p60 640×480p60/90	1080p30 720p60 640×480p60/90	1080p30 720p60 640×480p60/90
Linux integration	V4L2 driver available	V4L2 driver available	V4L2 driver available
C programming API	OpenMAX IL and others available	OpenMAX IL and others available	
Sensor	OmniVision OV5647	Sony IMX219	Sony IMX477
Sensor resolution	2592 × 1944 pixels	3280 × 2464 pixels	4056 x 3040 pixels
Sensor image area	3.76 × 2.74 mm	3.68 x 2.76 mm 4.6 mm diagonal	6.29 x 4.71 mm 7.9mm diagonal
Pixel size	1.4 µm × 1.4 µm	1.12 µm x 1.12 µm	1.55 µm x 1.55 µm
Optical size	1/4"	1/4"	
Focal length	3.60 mm +/- 0.01	3.04 mm	Depends on lens
Horizontal field of view	53.50 +/- 0.13 degrees	62.2 degrees	Depends on lens
Vertical field of view	41.41 +/- 0.11 degrees	48.8 degrees	Depends on lens
Focal ratio (F-Stop)	2.9	2.0	Depends on lens

Note:

- Camera Module V1 is obsolete
- Camera Module V2 will remain in production until at least January 2024

Installing a camera

The camera allows you to take pictures, videos as well as stream video.

Power your Pi off and connect the camera. Power on your Pi.

Now you need to enable camera support using the raspi-config program you will have used when you first set up your Raspberry Pi.

```
sudo raspi-config
```

Use the cursor keys to move to the camera option, and select 'enable'. On exiting raspi-config, it will ask to reboot. The enable option will ensure that on reboot the correct GPU firmware will be running with the camera driver and tuning, and the GPU memory split is sufficient to allow the camera to acquire enough memory to run correctly.

Note: For detailed information on raspistill, raspistillyuv and raspivid look in https://www.raspberrypi.org/documentation/raspbian/applications/camera.md

- **Taking a picture**

  To test that the camera is installed and working, try the following command:

  ```
  raspistill -v –n –t 1 -o test.jpg
  ```

  Take a picture, saved to the file test.jpg, without a preview whilst displaying various informational messages. Wait one secod bfore taking the picture

- **Taking a video**

  ```
  raspivid -o test.h264 -t 25000
  ```

  will record for a period of 25 seconds.

  The Pi captures video as a raw H264 video stream. This is great but many media players will refuse to play it unless it is “wrapped” in a suitable container format. Luckily it is easy to wrap the data and produce a standard MP4 video file which should play in most media players. To do this wrapping we will use MP4Box which you can install using :

  ```
  sudo apt-get –y install gpac
  ```

  Once installed you can then use the following command to wrap your H264 video data in an MP4 container file. This will allow most media players to play the video.

  ```
  MP4Box -fps 30 -add test.h264 test.mp4
  ```

  This will give you a nice video at 30 frames per second that should play in most modern media players.

- **Create a video from pictures (timelapse) on your Raspberry Pi**

  Raspberry Pi camera module is a good choice, but because we only use camera to capture still images at low frame rate, so a USB camera or webcam is also OK. Many webcams can connect directly into Raspberry Pi's USB port, otherwise use an powered USB hub. We will cover both the Raspberry Pi camera module and USB camera.

  Software preparation

  We need to take 2 steps to make a time-lapse video directly on Raspberry Pi:

  1. Capture still images at predetemined frame rate
  2. Render still images into video file using Raspberry Pi's hardware video encoder.

  We will use `raspistill` for Raspberry Pi camera module or `streamer` for USB camera. `raspistill` is already included in Raspbian. So we still need to install `streamer` if you want to use a UBS camera:

  ```
  sudo apt-get install –y streamer
  ```

  To do step 2, we won't use mencoder, ffmpeg, avconv... simply they don't support Raspberry Pi's GPU (yet). If you use those applications on Raspberry Pi, it will take a very long time to finish rendering video and nearly 100% CPU. The only solution to get hardware encoding is using OpenMAX plugin in gstreamer. The video codec for final time-lapse video will be H264.
  We need to add another repository for Raspbian to get OpenMAX plugin.

  ```
  sudo sh -c 'echo deb http://vontaene.de/raspbian-updates/ . main >>
  /etc/apt/sources.list'

  sudo apt-get update

  sudo apt-get install -y libgstreamer1.0-0 liborc-0.4-0 gir1.2-gst-plugins-
  base-1.0 gir1.2-gstreamer-1.0 gstreamer1.0-alsa gstreamer1.0-omx
  gstreamer1.0-plugins-bad gstreamer1.0-plugins-base gstreamer1.0-plugins-
  base-apps gstreamer1.0-plugins-good gstreamer1.0-plugins-ugly
  gstreamer1.0-pulseaudio gstreamer1.0-tools gstreamer1.0-x libgstreamer-
  plugins-bad1.0-0 libgstreamer-plugins-base1.0-0
  ```

  Let's check our installation result:

  ```
  gst-inspect-1.0 | grep omx
  ```

  You will see something like that:

  ```
  omx:  omxh264enc: OpenMAX H.264 Video Encoder
  omx:  omxvc1dec: OpenMAX WMV Video Decoder
  omx:  omxmjpegdec: OpenMAX MJPEG Video Decoder
  omx:  omxvp8dec: OpenMAX VP8 Video Decoder
  omx:  omxtheoradec: OpenMAX Theora Video Decoder
  omx:  omxh264dec: OpenMAX H.264 Video Decoder
  omx:  omxh263dec: OpenMAX H.263 Video Decoder
  omx:  omxmpeg4videodec: OpenMAX MPEG4 Video Decoder
  omx:  omxmpeg2videodec: OpenMAX MPEG2 Video Decoder
  ```

  Note the first line `omxh264enc`, that's the only encoder we will use

Using the tools

1. **Capture still images at predetemined frame rate**

- For Raspberry Pi camera module:
  As calculated in introduction, for 1 hour (3600000 ms) we take a shot every 2.5 seconds (2500 ms), so

```
raspistill –n –q 80 -w 1280 -h 720 -o timelapse%04d.jpeg -tl 2500 -t 3600000
```

  will create 1440 still images with a resolution of 1280 x 720 and a JPG quality of 80%. 100% is uncompressed and hence a big file while 80% is good comperssion with still good quality of picture but a fairly smaller file. We use %04d (4 digits integer) for the filenames producing the files timelapse0001.jpeg to timelapse1440.jpeg. A preview is not needed.

- For USB camera:

```
streamer -t 1440 -r 0.4 -s 1280x720 -o timelapse0000.jpeg
```

  where -t: number of frames, -r: frame rate (0.4 fps means 2.5 seconds per frame), -s: resolution will create 1440 still images, starting with filename timelapse0000.jpeg.

2. **Render still images into video file using Raspberry Pi's hardware video encoder**

Now you have still images. You don't need to transfer them to another computer. The Raspberry Pi GPU is strong enough to do so.
Before starting the conversion, flush all data to SD card:

```
sync
```

Now run gstreamer

```
gst-launch-1.0 multifilesrc location=timelapse%04d.jpeg index=1 caps="image/jpeg,framerate=24/1" ! jpegdec ! omxh264enc ! avimux ! filesink location=timelapse.avi
```

will encode images from timelapse0001.jpeg to the end into timelapse.avi. If you want to count from timelapse0000.jpeg, use index=0. You can change frame rate 24/1 to 25/1, 30/1 or your own frame rate. Your time-lapse video will be available after a few minutes!

Note:
- If you want a mp4 format, change extension from .avi to .mp4
- If for some reason, there are missing images in the series, gstreamer will freeze. So make sure all pictures are all there in a full serial series.
- Gstreamer might have problems when your picture files are (too) big. So when you encounter problems, reduce the resolution and/or the quality of the pictures taken to reduce the filesize per picture.

Again flush all data to SD card, to avoid data corruption:

```
sync
```

Now you can copy timelapse.avi to your computer through ssh with scp, or simply power off your Raspberry Pi and get your video file out from SD card.
You can put commands in step 1 and step 2 into a single .sh file to automate all the process.

- **Create a video from pictures (timelapse) on an external computer**

  Time-lapse photography is a technique whereby the frequency at which film frames are captured (the frame rate) is much lower than that used to view the sequence. When played at normal speed, time appears to be moving faster and thus lapsing. The idea to make a time-lapse video is simple: instead of recording or capturing frames at normal rate (24, 25 or 30 fps), you capture frames at much lower rate, and then assembly them into a video at normal rate. Time-lapse technique is usually used to film events which last for a long time in reality.
  As an example, suppose that we want to make a 1-minute time-lapse video for an event which lasts an hour in reality. We choose 24 fps for the video. So we need to capture 24 x 60 = 1440 frames (images). So the frame rate for capturing is 1440 frames/hour or 0.4 fps, i.e. we take an image after every 2.5 seconds.
  The resolution for the video is determined by our camera's capability and also by our intention. Suppose that it is 1280x720.

  So we have a from series of numerically sequential images such as img001.png, img002.png, img003.png, … of which we want to create a video (using the encoder libx264) etc.

  Note: It is important that all images in a series have the same size and format and that there are no images missing in the series.

  We will use the ffmpeg tool for this. Download from https://ffmpeg.org and install this tool on your computer, if not installed yet. Download the images files from your Raspberry Pi via eg FTP, or Win SCP or via copy them over via USB stick, …

  You can specify two frame rates:

  - The rate according to which the images are read, by setting -framerate before -i. The default for reading input is -framerate 25 which will be set if no -framerate is specified.

  - The output frame rate for the video stream by setting -r after -i or by using the fps filter. If you want the input and output frame rates to be the same, then just declare an input -framerate and the output will inherit the same value.

  By using a separate frame rate for the input and output you can control the duration at which each input is displayed and tell ffmpeg the frame rate you want for the output file. If the input -framerate is lower than the output -r then ffmpeg will duplicate frames to reach your desired output frame rate. If the input -framerate is higher than the output -r then ffmpeg will drop frames to reach your desired output frame rate.

  In this example each image will have a duration of 5 seconds (the inverse of 1/5 frames per second). The video stream will have a frame rate of 30 fps by duplicating the frames accordingly:

  ```
  ffmpeg -framerate 1/5 -i img%03d.png -c:v libx264 -r 30 -pix_fmt yuv420p
  out.mp4
  ```

  Starting with a specific image

  For example if you want to start with img126.png then use the -start_number option:

  ```
  ffmpeg -framerate 1/5 -start_number 126 -i img%03d.png -c:v libx264 -r 30
  -pix_fmt yuv420p out.mp4
  ```

If your video does not show the frames correctly

If you encounter problems, such as the first image is skipped or only shows for one frame, then use the fps video filter instead of -r for the output framerate (see ticket:1578 and ticket:2674 / ticket:3164 for more info):

```
ffmpeg -framerate 1/5 -i img%03d.png -c:v libx264 -vf fps=25 -pix_fmt
yuv420p out.mp4
```

Alternatively the format video filter can be added to the filterchain to replace -pix_fmt yuv420p. The advantage to this method is that you can control which filter goes first:

```
ffmpeg -framerate 1/5 -i img%03d.png -c:v libx264 -vf
"fps=25,format=yuv420p" out.mp4
```

Color space conversion and chroma sub-sampling

By default when using libx264, and depending on your input, ffmpeg will attempt to avoid color subsampling. Technically this is preferred, but unfortunately almost all video players, excluding FFmpeg based players, and many online video services only support the YUV color space with 4:2:0 chroma subsampling. Using the options `-pix_fmt yuv420p` or `-vf format=yuv420p` will maximize compatibility.

Using a glob pattern

ffmpeg also supports bash-style globbing (* represents any number of any characters). This is useful if your images are sequential but not necessarily in a numerically sequential order as in the previous examples.

```
ffmpeg -framerate 1 -pattern_type glob -i '*.jpg' -c:v libx264 out.mp4
```

For PNG images:

```
ffmpeg -framerate 1 -pattern_type glob -i '*.png' -c:v libx264 -pix_fmt
yuv420p out.mp4
```

Using a single image as an input

If you want to create a video out of just one image, this will do (output video duration is set to 30 seconds with -t 30):

```
ffmpeg -loop 1 -i img.png -c:v libx264 -t 30 -pix_fmt yuv420p out.mp4
```

Adding audio

If you want to add audio (e.g. audio.wav) to one "poster" image, you need -shortest to tell it to stop after the audio stream is finished. The internal AAC encoder is used in this example, but you can use any other supported AAC encoder as well:

```
ffmpeg -loop 1 -i img.jpg -i audio.wav -c:v libx264 -c:a aac -strict
experimental -b:a 192k -shortest out.mp4
```

If your audio file is using a codec that the output container supports (e.g. MP3 audio in AVI or M4A/AAC audio in MP4), you can copy it instead of re-encoding, which will preserve the audio quality:

```
ffmpeg -loop 1 -i img.jpg -i audio.m4a -c:v libx264 -c:a copy -shortest
out.mp4
```

- **Stream a stream**

```
cd ~
sudo apt-get –y install cmake
sudo modprobe –v bcm2835-v4l2
sudo git clone https://github.com/mpromonet/h264_v4l2_rtspserver.git
cd h264_v4l2_rtspserver
sudo cmake .   --> you might need to run this command 2 times
sudo make install
h264_v4l2_rtspserver
```

Next you should be able to access to the RTSP stream using VLC on any device by opening the stream `rtsp://<ip.of.rasp.pi>:8554/unicast. Eg.`

```
vlc rtsp://<raspberry>:8554/unicast
```

Or open VLC player, click on Media -> Open Network Stream -> Network. Enter the network URL:

```
rtsp://<raspberry>:8554/unicast
```

Note: To see all full command line option, type

```
h264_v4l2_rtspserver –?
```

- **To LED ot not to LED**

The Pi camera module (at least the first version of it) includes a red LED in one corner of the PCB. This lights up when the camera is active. It's really useful in giving a visual indication that the camera is doing something and most of the time you will be glad it is there. However there are a number of reasons you might wish it wasn't, like

- It can cause reflections on objects you are trying to photograph giving them a red glow.
- For nature photography it scares animals.
- For security applications it may draw unnecessary attention to the device.
- It consumes power.

To disable the red LED edit the `config.txt` file

```
sudo nano /boot/config.txt
```

and add the following:

```
disable_camera_led=1
```

When you have finished press [Ctrl] + X. This will ask if you want to save the modified files. Press 'Y' and then hit [Return] to save the file with the same name.

Reboot your Pi

```
sudo reboot
```

To enable the light again you can either remove the line you added above or you can change it to

```
disable_camera_led=0
```

Reboot the Pi and you will have your camera light back.

- ***Python modules for Pi Camera***

  Besides the `raspistill` and `raspivid` tools, there are also Python modules available to handle the Pi Camera. Just execute the command below to get them for sure on your Pi

```
sudo apt-get update
sudo apt-get –y install python-picamera python3-picamera
```

Add face recognition

It's hard to comprehend how far machine learning has come in the past few years.

You can now use a sub-£50 computer to reliably recognise someone's face with surprising accuracy. Although this kind of computing power is normally out of reach of microcontrollers, adding a Raspberry Pi computer to your project with the new High Quality Camera opens up a range of possibilities. From simple alerting applications ('Mum's arrived home!'), to dynamically adjusting settings based on the person using the project, there's a lot of fun to be had.

Prepare your raspberry pi
For face recognition to work well, we're going to need some horsepower, so we recommend a minimum of Raspberry Pi 3B+, ideally a Raspberry Pi 4. The extra memory will make all the difference. To keep as much resource as possible available for our project, we've gone for a Raspberry Pi OS Lite installation with no desktop. Make sure you're on the network, have set a new password, enabled SSH if you need to, and updated everything with

```
sudo apt -y update
sudo apt -y full-upgrade
```

Finally, go into settings by running

```
sudo raspi-config
```

and enable the camera in 'Interfacing Options'.

Attach the camera
This project will work well with the original Raspberry Pi Camera, but the new official HQ Camera will give you much better results. Be sure to connect the camera to your Raspberry Pi 4 with the power off. Connect the ribbon cable as instructed. Once installed, boot up your Raspberry Pi 4 and test the camera is working. From the command line, run the following:

```
raspivid -o test.h264 -t 10000
```

This will record ten seconds of video to your microSD card. If you have an HDMI cable plugged in, you'll see what the camera can see in real-time. Take some time to make sure the focus is correct before proceeding.

Note: Don't worry if you don't have a Raspberry Pi High Quality Camera – you can use the original or any compatible USB camera

Install dependencies
The facial recognition library we are using is one that has been maintained for many years by Adam Geitgey (github.com/ageitgey/face_recognition).
It contains many examples, including Python 3 bindings to make it really simple to build your own facial recognition applications. What is not so easy is the number of dependencies that need to be installed first.

```
sudo apt install build-essential cmake gfortran git wget curl \
    graphicsmagick libgraphicsmagick1-dev libatlas-base-dev \
    libavcodec-dev libavformat-dev libboost-all-dev \
    libgtk2.0-dev libjpeg-dev liblapack-dev \
    libswscale-dev pkg-config python3-dev \
    python3-numpy python3-pip zip python3-picamera
```

Update

```
sudo pip3 install --upgrade picamera[array]
```

Increase the swap file size so we can build dlib

```
sudo nano /etc/dphys-swapfile
```

Find `CONF_SWAPSIZE` and change its value from 100 to 1024. Save and exit then run this command:

```
sudo /etc/init.d/dphys-swapfile restart
```

Build and install dlib

```
cd
git clone -b 'v19.6' --single-branch https://github.com/davisking/dlib.git
cd ./dlib
sudo python3 setup.py install --compiler-flags "-mfpu=neon"
```

This step will take a while to complete on a Raspberry Pi 4, and significantly longer on a Model 3 or earlier. (Raspberry Pi 4 took about 30 minutes)

Revert the swap size

```
sudo nano /etc/dphys-swapfile
```

Find `CONF_SWAPSIZE` and change its value from 1024 to 100. Save and exit then run this command:

```
sudo /etc/init.d/dphys-swapfile restart
```

Install the libraries
Now that we have everything in place, we can install Adam's applications and Python bindings with a simple, single command:

```
sudo pip3 install face_recognition
```

Once installed, there are some examples we can download to try everything out.

```
cd
git clone --single-branch https://github.com/
ageitgey/face_recognition.git
```

In this repository is a range of examples showing the different ways the software can be used, including live video recognition. Feel free to explore and remix.

Example images
The examples come with a training image of Barack Obama. To run the example:

```
cd ./face_recognition/examples
python3 facerec_on_raspberry_pi.py
```

On your smartphone, find an image of Obama using your favourite search engine and point it at the camera. Providing focus and light are good you will see: "I see someone named Barack Obama!"
If you see a message saying it can't recognise the face, then try a different image or try to improve the lighting if you can. Also, check the focus for the camera and make sure the distance between the image and camera is correct.

Training time

The final step is to start recognising your own faces. Create a directory and, in it, place some good-quality passport-style photos of yourself or those you want to recognise. You can then edit the `facerec_on_raspberry_pi.py` script to use those files instead. You've now got a robust prototype of face recognition. This is just the beginning. These libraries can also identify 'generic' faces, meaning it can detect whether a person is there or not, and identify features such as the eyes, nose, and mouth. There's a world of possibilities available, starting with these simple scripts.

```
# This is a demo of running face recognition on a Raspberry Pi.
# This program will print out the names of anyone it recognizes to the console.

# To run this, you need a Raspberry Pi 2 (or greater) with face_recognition and
# the picamera[array] module installed.
# You can follow this installation instructions to get your RPi set up:
# https://gist.github.com/ageitgey/1ac8dbe8572f3f533df6269dab35df65

import face_recognition
import picamera
import numpy as np

# Get a reference to the Raspberry Pi camera.
# If this fails, make sure you have a camera connected to the RPi and that you
# enabled your camera in raspi-config and rebooted first.
camera = picamera.PiCamera()
camera.resolution = (320, 240)
output = np.empty((240, 320, 3), dtype=np.uint8)

# Load a sample picture and learn how to recognize it.
print("Loading known face image(s)")
obama_image = face_recognition.load_image_file("obama_small.jpg")
obama_face_encoding = face_recognition.face_encodings(obama_image)[0]

# Initialize some variables
face_locations = []
face_encodings = []

while True:
    print("Capturing image.")
    # Grab a single frame of video from the RPi camera as a numpy array
    camera.capture(output, format="rgb")

    # Find all the faces and face encodings in the current frame of video
    face_locations = face_recognition.face_locations(output)
    print("Found {} faces in image.".format(len(face_locations)))
    face_encodings = face_recognition.face_encodings(output, face_locations)

    # Loop over each face found in the frame to see if it's someone we know.
    for face_encoding in face_encodings:
        # See if the face is a match for the known face(s)
        match = face_recognition.compare_faces([obama_face_encoding], face_encoding)
        name = "<Unknown Person>"

        if match[0]:
            name = "Barack Obama"

        print("I see someone named {}!".format(name))
```

Appendix

Using ffmpeg

Downloading & Compiling Libx264
In order to compile it from source with h264 support, first install the h264 libs:

```
cd /usr/src
git clone --depth 1 https://code.videolan.org/videolan/x264.git
cd x264
./configure --host=arm-unknown-linux-gnueabi --enable-static --disable-opencl
make -j4
sudo make install
```

Use `make` instead of `make -j4` if you only have 1 core in your Raspberry Pi. `-j4` indicates that you have 4 cores and that `ffmpeg` will be able to use all 4 cores

Downloading & Compiling FFMPEG
I just want video output, so I skipped the audio requirements (or else you should install these now). So I continued with `ffmpeg` itself:

```
cd /usr/src
git clone git://source.ffmpeg.org/ffmpeg.git
cd ffmpeg
sudo ./configure --arch=armel --target-os=linux --enable-gpl --enable-libx264
--enable-nonfree
make -j4
```

Now you have to wait a long time for it to finish. When finished,

```
sudo make install
```

Finishing up
Now that `ffmpeg` has been compiled and installed successfully, we will need to reboot to make sure that everything is working. Type in;

```
sudo reboot
```

When the Pi boots up, type in

```
ffmpeg
```

and you should see an output similar to that of what I get below;

```
ffmpeg version N-81410-g4d7d748 Copyright (c) 2000-2016 the FFmpeg developers
  built with gcc 4.9.2 (Raspbian 4.9.2-10)
  configuration: --arch=armel --target-os=linux --enable-gpl --enable-libx264 --
enable-nonfree
  libavutil      55. 29.100 / 55. 29.100
  libavcodec     57. 54.100 / 57. 54.100
  libavformat    57. 47.101 / 57. 47.101
  libavdevice    57.  0.102 / 57.  0.102
  libavfilter     6. 52.100 /  6. 52.100
  libswscale      4.  1.100 /  4.  1.100
  libswresample   2.  1.100 /  2.  1.100
  libpostproc    54.  0.100 / 54.  0.100
Hyper fast Audio and Video encoder
usage: ffmpeg [options] [[infile options] -i infile]... {[outfile options]
outfile}...

Use -h to get full help or, even better, run 'man ffmpeg'
```

And that's it! You have now successfully installed FFMPEG on your Raspberry Pi!

Making a time lapse with ffmpeg
If you have a series of JPEG which are taken over a period of time, you can use the following command to create a timelapse video.

```
ffmpeg -r 1 -i images-%06d.jpg -r 30 -s hd720 -vcodec libx264 output.mp4 -y -v fatal
```

Let me go through this command

In general, any options before `-i` denote the input file parameters. For this, we have `-r 1 -i images-%02d.jpg`

`-r` is to denote the input frame rate. This is important and is a common mistake made by user. If you do not put -r, it will default -r as 25. You need to define a rate of the input files that match with the output frame rate. `-r 1` means 1 image per second. So if you have 100 images, your time lapse will have a length of 100 seconds

NB: To make your timelapse "faster", change `-r 1` to `-r 30` or any number you wish

`-i` is to set your input file names. %06 means that `ffmpeg` will take images-00000.jpg to images-99999.jpg as input consideration.

Next, any option after -i denote the output file parameters. For this, we have `-r 30 -s hd7200 -vcodec libx264 output.mp4`

`-r` is to denote the output frame rate.

`-s` is to set the output frame size to hd1080 or hd720

`-vcodec` denote the codec to be used to encode the output file. libx264 is the standard encoder for H.264 encoding in FFMPEG

`output.mp4` is the output name

And at the end we can set some general parameters

`-y` is to overwrite an existing file output.mp4 should it exist

`-v fatal` is to only log fatal errors to the screen. Options are `quiet, panic, fatal, error, warning, info, verbose, debug, trace`

NB: When utilising FFMPEG, you will see that it may bring the processor usage to 100% on your Pi. This will cause the SoC to heat up rapidly unless you have a heatsink or some other form of cooling. At 100% CPU utilisation, the temperature without a heatsink can reach up to 80° Celsius!